Kotlin - Arrays

Arrays are used to store multiple items of the same data-type in a single variable, such as an integer or string under a single variable name.

For example, if we need to store names of 1000 employees, then instead of creating 1000 different string variables, we can simply define an array of string whose capacity will be 1000.

Like any other modern programming languages, Kotlin also supports arrays and provide a wide range of array properties and support functions to manipulate arrays.

Creating Arrays in Kotlin

To create an array in Kotlin, we use the **arrayOf()** function, and place the values in a comma-separated list inside it:

```
val fruits = arrayOf("Apple", "Mango", "Banana", "Orange")
```

Optionally we can provide a data type as follows:

```
val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange")
```

Alternatively, the **arrayOfNulls()** function can be used to create an array of a given size filled with null elements.

Primitive type Arrays

Kotlin also has some built-in factory methods to create arrays of primitive data types. For example, the factory method to create an integer array is:

```
val num = intArrayOf(1, 2, 3, 4)
```

Other factory methods available for creating arrays:

- byteArrayOf()
- charArrayOf()
- shortArrayOf()

- longArrayOf()

Get and Set the Elements of an Array

We can access an array element by using the index number inside square brackets. Kotlin array index starts with zero (0). So if you want to access 4th element of the array then you will need to give 3 as the index.

Example

```
fun main(args: Array<String>) {
   val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange")

   println( fruits [0])
   println( fruits [3])

}
```

When you run the above Kotlin program, it will generate the following output:

```
Apple
Orange
```

Kotlin also provides **get()** and **set()** member functions to get and set the value at a particular index. Check the following example:

Example

```
fun main(args: Array<String>) {
   val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange")

   println( fruits.get(0))
   println( fruits.get(3))

   // Set the value at 3rd index
   fruits.set(3, "Guava")
   println( fruits.get(3))
}
```

When you run the above Kotlin program, it will generate the following output:

```
Apple
Orange
Guava
```

Kotlin Array Length

Kotlin provides array property called **size** which returns the size i.e. length of the array.

Example

```
fun main(args: Array<String>) {
   val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange")

   println( "Size of fruits array " + fruits.size )

}
```

When you run the above Kotlin program, it will generate the following output:

```
Size of fruits array 4
```

We can also use **count()** member function to get the size of the array.

Loop Through an Array

We can use **for** loop to loop through an array.

Example

```
fun main(args: Array<String>) {
   val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange")

   for( item in fruits ){
      println( item )
   }

}
```

When you run the above Kotlin program, it will generate the following output:

```
Apple
Mango
Banana
Orange
```

Check if an Element Exists

We can use the **in** operator alongwith **if...else** to check if an element exists in an array.

Example

```
fun main(args: Array<String>) {
   val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange")

   if ("Mango" in fruits){
      println( "Mango exists in fruits" )
   }else{
      println( "Mango does not exist in fruits" )
   }
}
```

When you run the above Kotlin program, it will generate the following output:

```
Mango exists in fruits
```

Distinct Values from Array

Kotlin allows to store duplicate values in an array, but same time you can get a set of distinct values stored in the array using **distinct()** member function.

Example

```
fun main(args: Array<String>) {
   val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange", "Apple")

   val distinct = fruits.distinct()
   for( item in distinct ){
      println( item )
   }
}
```

When you run the above Kotlin program, it will generate the following output:

```
Apple
Mango
Banana
Orange
```

Dropping Elements from Array

We can use **drop()** or **dropLast()** member functions to drop elements from the beginning or from the last respectively.

Example

```
fun main(args: Array<String>) {
   val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange", "Apple")

   val result = fruits.drop(2) // drops first two elements.
   for( item in result ){
      println( item )
   }
}
```

When you run the above Kotlin program, it will generate the following output:

```
Banana
Orange
Apple
```

Checking an Empty Array

We can use **isEmpty()** member function to check if an array is empty or not. This function returns true if the array is empty.

Example

```
fun main(args: Array<String>) {
   val fruits = arrayOf<String>()
   println( "Array is empty : " + fruits.isEmpty())

}
```

When you run the above Kotlin program, it will generate the following output:

```
"Array is empty : true
```

Quiz Time (Interview & Exams Preparation)

Q 1 - Which of the following is true about Kotlin Arrays?

A - Kotlin arrays are capable to store all data types.

B - Kotlin arrays can be created using either arrayOf() or arrayOfNulls() functions.

C - Kotlin provides a rich set of properties and functions to manipulate arrays.

D - All of the above

Q 2 - What will be the output of the following code segment?

```
fun main(args: Array<String>) {
   val fruits = arrayOf<String>("Apple", "Mango", "Banana", "Orange")

   println( fruits [2])

}
```

A - Apple

B - Mango

C - Banana

D - None of the above

Q 3 - How to get the size of a Kotlin array?

A - Using length() function

B - Using size property

C - Using count() function

D - B and C both